Exercise 1: B+ Trees

Consider the following B+ tree index of order 1:

1. Circle all nodes in the above figure that must be fetched to satisfy the query "Get all records with search key greater than or equal to 7 and less than 15".
2. Assume we modify the B+ tree by adding the following keys in the following order: 20, 27, 18, 30, 19. In the answer-boxes below, each row refers to a key being inserted in order, and each column asks if the insertion of that key results in a split of particular nodes. Place a check mark (✓) in each box whose answer is “Yes”. Blank boxes will be interpreted as “No”.

Answer:

<table>
<thead>
<tr>
<th>Key</th>
<th>Leaf Node Split?</th>
<th>Non-Leaf Split?</th>
<th>Root Split?</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Suppose we were to insert all integers in the range 26 to 4112 inclusive (i.e. 26, 27, 28 .., 4111, 4112) into the tree in part (a), one at a time. At most how many levels would the resulting B+-tree have? (Hint: You should not need to draw a B+-tree to figure this out!)

Answer:

If there are K search-key values in the file, the path is no longer than \(\lceil \log_{\frac{n}{2}}(K) \rceil\).

- \(n = 3\)
- Number of search keys: \(K = 9 + (4112 - 26) + 1 = 4096\)
- Number of levels= \(\lceil \log_{\frac{3}{2}}(4096) \rceil\).
Exercise 2: Extendible Hashing

Answer the following questions about Extendible Hashing:

1. Explain why local depth and global depth are needed.

**Answer:** Extendible hashing allows the size of the directory to increase and decrease depending on the number and variety of inserts and deletes. Once the directory size changes, the hash function applied to the search key value should also change. So there should be some information in the index as to which hash function is to be applied. This information is provided by the *global depth*.

An increase in the directory size doesn’t cause the creation of new buckets for each new directory entry. All the new directory entries except one share buckets with the old directory entries. Whenever a bucket which is being shared by two or more directory entries is to be split the directory size need not be doubled.

This means for each bucket we need to know whether it is being shared by two or more directory entries. This information is provided by the *local depth* of the bucket. The same information can be obtained by a scan of the directory, but this is costlier.

2. After an insertion that causes the directory size to double, how many buckets have exactly one directory entry pointing to them? If an entry is then deleted from one of these buckets, what happens to the directory size? Explain your answers briefly.

**Answer:** Exactly two directory entries have only one directory entry pointing to them after a doubling of the directory size. This is because when the directory is doubled, one of the buckets must have split causing a directory entry to point to each of these two new buckets.

If an entry is then deleted from one of these buckets, a merge may occur, but this depends on the deletion algorithm. If we try to merge two buckets only when a bucket becomes empty, then it is not necessary that the directory size decrease after the deletion that was considered in the question. However, if we try to merge two buckets whenever it is possible to do so then the directory size decreases after the deletion.
3. Does Extendible Hashing guarantee at most one disk access to retrieve a record with a given key value?

**Answer:** No "minimum disk access" guarantee is provided by extendible hashing. If the directory is not already in memory it needs to be fetched from the disk which may require more than one disk access depending upon the size of the directory. Then the required bucket has to be brought into the memory. Also, if alternatives 2 and 3 are followed for storing the data entries in the index then another disk access is possibly required for fetching the actual data record.

4. Does doubling the directory require us to examine all buckets with local depth equal to global depth?

**Answer:** No. Since we are using extendible hashing, only the local depth of the bucket being split needs be examined.
Exercise 3: Query Processing

Consider a database with two relations, R1(a, b) and R2(c, d). The tuples of both R1 and R2 are ordered on disk. R2.c is a foreign key that references R1.b. R1 has 1,000,000 tuples of size 80 bytes each; 50 R1 tuples fit on a disk page. R2 has 4000 tuples of size 200 bytes each; 20 R2 tuples fit on a page. There is an B+Tree index on R1 with order d=2 on the key <b>. Suppose the DBMS has 102 buffer pages (M).

We wish to evaluate the following simple SQL query:

SELECT R1.a, R1.b, R2.d FROM R1, R2 WHERE R1.b = R2.c

1. Using block nested loops join, how many disk accesses are required?
2. Using index nested loops join, how many disk accesses are required?

Answer:

- T(R1) = 1000000
- B(R1) = 1000000/50 = 20000
- T(R2) = 4000
- B(R2) = 4000/20 = 200

- M = 102
- B+Tree order = 2
- The max number of search key in the node = 4
- The number of pointer in the node = 5

1. BNLJ (without considering M): B(R2)*B(R1) + B(R2) = 200*2000 + 200
2. BNLJ (consider M): B(R2) + \( B(R2) \times \frac{B(S)}{M-2} \) = 200 + \( \frac{200 \times 20000}{100} \)
3. INLJ: Outer relation: R2

Inner is the index of R1.

The cost is = B(R2) + T(R2) * (h + c)
where h denotes the height of the index. h = \( \lceil \log_{\frac{d}{2}}(1000000) \rceil \).
If b is the primary key of R1 c =1; if b is nonkey than c is the number of blocks containing records with the specified search key.